Generating custom sitemap with Solr

This is a summary of how to install and run Solr in Magnolia. The information provided in this entry has been retrieved from several places and people: Magnolia documentation, Milan Divilek and Jonathan Ayala hints, Solr official documentation, a Mykong site entry and a bit of self investigation. As an overview, we will follow 4 main steps:

1. Install and configure Solr
2. Activate and run Crawler
3. Create java class to query and provide Solr search results
4. Create FTL file for invoking Solr search and rendering the result into a XML file

- Install and configure Solr -

For this tutorial I'm using magnolia 5.7.2, EE (generated via mvn org.apache.maven.plugins:maven-archetype-plugin:2.4:generate -DarchetypeCatalog=https://nexus.magnolia-cms.com/content/groups/public/)

I included the following dependencies in my pom:

```
<dependency>
  <groupId>info.magnolia.solr</groupId>
  <artifactId>magnolia-content-indexer</artifactId>
  <version>5.2.1</version>
</dependency>
<dependency>
  <groupId>info.magnolia.solr</groupId>
  <artifactId>magnolia-solr-search-provider</artifactId>
  <version>5.2.1</version>
</dependency>
<dependency>
  <groupId>info.magnolia.solr</groupId>
  <artifactId>magnolia-solr-workbench</artifactId>
  <version>5.2.1</version>
</dependency>
```

Now, we can mount the project in our server, start and install Magnolia (no restart will be needed beyond this point).

Download and install Solr from here http://lucene.apache.org/solr/mirrors-solr-latest-redir.html. For your comfort, I would recommend you to set SOLR_HOME var and add it to your PATH in order to run Solr command without using the absolute path.

```
#set Solr
export SOLR_HOME=/usr/local/solr-7.3.0
#set PATH
export PATH=$PATH:$SOLR_HOME/bin
```

As explained here, we must create a new Magnolia config set by duplicating the $SOLR_HOME/server/solr/configsets/_default folder and naming it to magnolia_data_driven_schema_configs.

In this new configuration set you need to create or modify two files, solrconfig.xml and managed-schema (be aware that different Solr versions may require different content in the Solr configuration files, this wiki entry is expecting you to run Magnolia 5.7.2).

Start Solr using the command

```
> ./solr start
```

You will get something like:

```
Waiting up to 180 seconds to see Solr running on port 8983 [\]
Started Solr server on port 8983 (pid=20929). Happy searching!
```

Now we can check Solr status by using the following command:

```
> ./solr status
```

```
Found 1 Solr nodes:

Solr process 20929 running on port 8983
{
  "solr_home":"/usr/local/solr-7.3.0",
  "version":"7.3.0 98a6b3d642928b1ac9076c6c5a369472581f7633 - woody - 2018-03-28 14:37:45",
  "startTime":"2019-04-11T11:51:41.545Z",
  "uptime":"0 days, 0 hours, 4 minutes, 39 seconds",
  "memory":"53 MB (%10.8) of 490.7 MB"}
```

As the output says, Solr is running on port 8963, so we can go to http://localhost:8983/solr/#/ and check it the admin dashboard is working.

Next step is create Magnolia core. A core is a running instance of a Lucene index along with all the Solr configuration required to use it. For this purpose, we must execute the command:

```
> ./solr create_core -c magnolia -d magnolia_data_driven_schema_configs
```

If everything went ok, you will see the message: *Created new core 'magnolia'* and you will be able of accessing the url http://localhost:8983/solr/#/magnolia/query (or if you rather, just refresh the http://localhost:8983/solr/#/ and in the core selector you must be able of selecting the new core we just created).

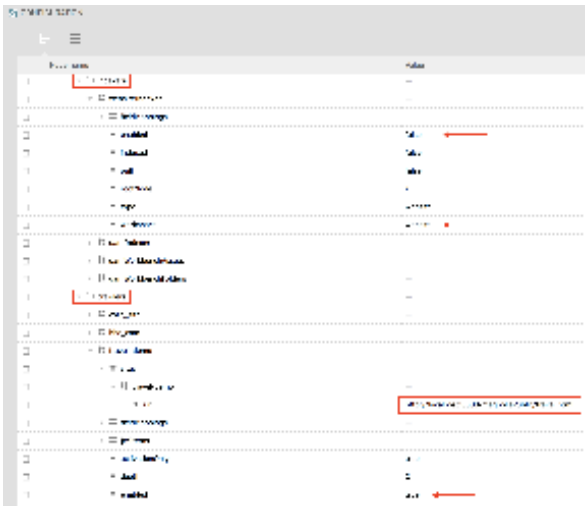
- Activate and run Crawler -

By injecting the dependency 'magnolia-solr-search-provider', we will have a *solr-search-provider* module preconfigured that will fit our needs.

Node name	Value
solr-search-provider	
config	
solrClientConfigs	
default	
allowCompression	false
baseURL	http://localhost:8983/solr/magnolia
connectionTimeout	100
soTimeout	1,000

For this tutorial we will use crawlers, the difference between Index and Crawl is quite simple: indexing is a technic with tells Googlebot which pages must be indexed and are elegible to be showed in SERPs. On the other hand, crawling allows Googlebot to go inside a page and analyze it (eg, javascript files, images...). It's possible to enable crawling on a site and use robots (noIndex tag) for letting Google know he must not go inside a document (although he will index the entry to that page but not the page as itself).

So, we are deactivating Index and activating crawling (as said above, they aren't mutually inclusive). For that taks, we go to *content-indexer/config/crawlers/* and add our site (e.g, trave_demo site). In *fieldMappings* we add the elements we want to retrieve from the site (in the example, image tag will only work if the images are embebbed in the app, if they are linked to dam workspace it wont work, we will need to retrieve the assets linked to the JCR elements). We set webIndexer to false and crawlers to true as explains the following image:



- Create java class to query and provide Solr search results -

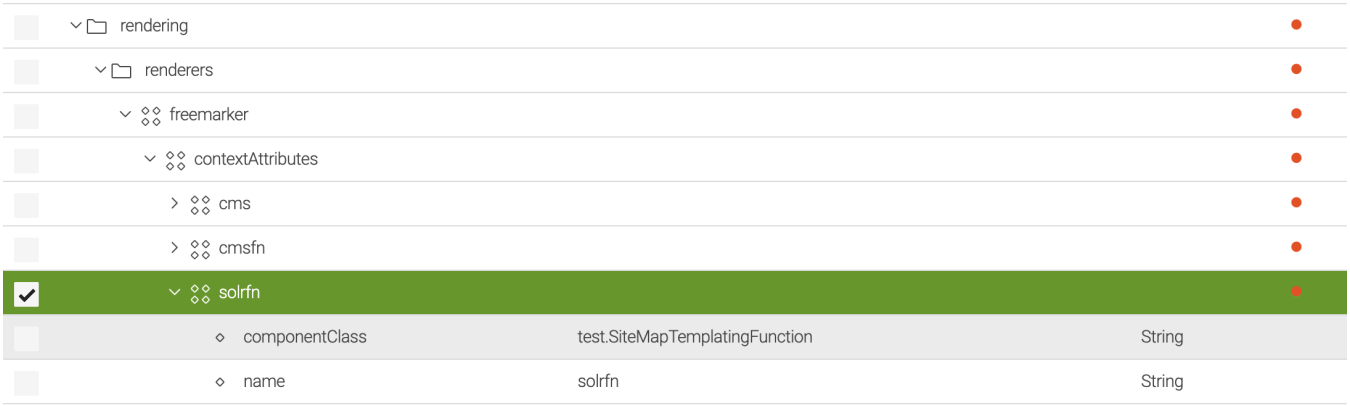
This java class that will implement the *getAllSolrIndexes*() method (courtesy of Mikaël Geljić) that will be invoked in our FTL from a custom templating function. The purpose of this class is provide the Solr query Search and return the matching elements. This java class is like a model class and it´s included into this sample module:

```
public class SitemapTemplatingFunction {

    private static final Logger log = LoggerFactory.getLogger(SitemapTemplatingFunction.class);

    public Collection<SolrDocument> getAllSolrIndexes() throws SolrServerException, IOException {
        return getAllSolrIndexes(null, null);
    }

    public Collection<SolrDocument> getAllSolrIndexes(String type) throws SolrServerException, IOException {
        return getAllSolrIndexes(type, null);
    }

    public Collection<SolrDocument> getAllSolrIndexes(String type, String... fields) throws
SolrServerException, IOException {
        MagnoliaSolrBridge magnoliaSolrBridge = Components.getComponent(MagnoliaSolrBridge.class);
        SolrClient solrClient = magnoliaSolrBridge.getSolrClient();

        SolrQuery solrQuery = new SolrQuery();
        solrQuery.setQuery("*");
        solrQuery.setStart(0);
        solrQuery.setSort("url", SolrQuery.ORDER.asc);

        // perform query with zero rows, to resolve number of results
        solrQuery.setRows(0);
        QueryResponse queryResponseNumberOfResults = solrClient.query(solrQuery);

        // resolve number of results
        int numberOfResults = Long.valueOf(queryResponseNumberOfResults.getResults().getNumFound()).intValue();
        log.debug("Found {} document(s)", numberOfResults);
        solrQuery.setRows(numberOfResults);

        if (fields != null && fields.length > 0) {
            solrQuery.setFields(fields);
        }
        solrQuery.addField("url");

        if (type != null) {
            solrQuery.addFilterQuery("type:" + type);
        }

        // perform query to get all result
        QueryResponse queryResponse = solrClient.query(solrQuery);
        return queryResponse.getResults();
    }
}
```

- Create FTL file -

As said above, we need to define a templating function that will be called by the FTL script and will invoke the getAllSolrIndexes() method from the java class created before. This file will also render the result into a XML file (the elements we are allowed to add to this generation are the same elements we place in the fieldsMapping configuration).

For this example I'm defining a *solrfn* function that will be linked to our SitemapTemplatingFunction java class:



For creating your FTL script please just copy the following snippet that invokes the getAllSolrIndexes method and generates the sitemap definition:

> This document describes the XML schema for the Sitemap protocol.

```
<?xml version="1.0" encoding="UTF-8"?>
[#compress]<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
[#assign searchResults = solrfn.getAllSolrIndexes() /]
[#list searchResults as item]

[#assign slashCount = item["url"]?remove_ending("/")?split("/")]
[#assign priority = (13 - slashCount?size)/10]

<url>
<changefreq>weekly</changefreq>
<loc>${item["url"]?html}</loc>
<priority>${priority}</priority>
</url>
[/#list]
</urlset>
[/#compress]
```

Now you can access to the absolute URL of the FTL script to test the sitemap generation. You can also go to http://localhost:8983/solr/#/magnolia/query and execute a Search query on Magnolia core to see if all the expected elements are getting found and shown.